ASSIGNMENT 2

11. Container With Most Water You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

class Solution:

def maxArea(self, height: List[int]) -> int:

p1 = 0

p2 = len(height) - 1

max\_area = 0

while p1 != p2:

if height[p1] > height[p2]:

area = height[p2] \* (p2 - p1)

p2 -= 1

else:

area = height[p1] \* (p2 - p1)

p1 += 1

if area > max\_area: max\_area = area

return max\_area

12. Integer to Roman Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given an integer, convert it to a roman numeral

def printRoman(number):

num = [1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000]

sym = ["I", "IV", "V", "IX", "X", "XL", "L", "XC", "C", "CD", "D", "CM", "M"]

i = 12

while number:

div = number // num[i]

number %= num[i]

while div:

print(sym[i], end="")

div -= 1

i -= 1

if \_\_name\_\_ == "\_\_main\_\_":

number = 3549

print("Roman value is:", end=" ")

printRoman(number)

13. Roman to Integer Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given a roman numeral, convert it to an integer.

def value(r):

if (r == 'I'):

return 1

if (r == 'V'):

return 5

if (r == 'X'):

return 10

if (r == 'L'):

return 50

if (r == 'C'):

return 100

if (r == 'D'):

return 500

if (r == 'M'):

return 1000

return -1

def romanToDecimal(str):

res = 0

i = 0

while (i < len(str)):

s1 = value(str[i])

if (i + 1 < len(str)):

s2 = value(str[i + 1])

if (s1 >= s2):

res = res + s1

i = i + 1

else:

res = res + s2 - s1

i = i + 2

else:

res = res + s1

i = i + 1

return res

print("Integer form of Roman Numeral is"),

print(romanToDecimal("MCMIV"))

14. Longest Common Prefix Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string ""

class Solution:

def longestCommonPrefix(self, strs: List[str]) -> str:

pref = strs[0]

pref\_len = len(pref)

for s in strs[1:]:

while pref != s[0:pref\_len]:

pref\_len -= 1

if pref\_len == 0:

return ""

pref = pref[0:pref\_len]

return pref

15. 3Sum Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets. Example 1: Input: nums = [-1,0,1,2,-1,-4] Output: [[-1,-1,2],[-1,0,1]] Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter.

class Solution:

def threeSum(self, nums: List[int]) -> List[List[int]]:

nums.sort()

ans = []

n = len(nums)

i = 0

while i < n:

if nums[i] > 0: break # Since arr[i] <= arr[l] <= arr[r], if arr[i] > 0 then sum=arr[i]+arr[l]+arr[r] > 0

l = i + 1

r = n - 1

while l < r:

sum3 = nums[i] + nums[l] + nums[r]

if sum3 == 0:

ans.append([nums[i], nums[l], nums[r]])

while l+1 < n and nums[l+1] == nums[l]: l += 1 # Skip duplicates nums[l]

l += 1

r -= 1

elif sum3 < 0: l += 1

else: r -= 1

while i+1 < n and nums[i+1] == nums[i]: i += 1 # Skip duplicates nums[i]

i += 1

return ans

16. 3Sum Closest Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution

class Solution:

def threeSumClosest(self, nums: List[int], target: int) -> int:

nums.sort()

n = len(nums)

ans = nums[0] + nums[1] + nums[2]

for i in range(n-2):

l, r = i + 1, n - 1

while l < r:

sum3 = nums[i] + nums[l] + nums[r]

if abs(ans - target) > abs(sum3 - target):

ans = sum3

if sum3 == target: return target

if sum3 > target:

r -= 1

else:

l += 1

return ans

17. Letter Combinations of a Phone Number Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

class Solution:

def letterCombinations(self, digits: str) -> List[str]:

if not digits:

return []

digit\_to\_letters = {

'2': 'abc',

'3': 'def',

'4': 'ghi',

'5': 'jkl',

'6': 'mno',

'7': 'pqrs',

'8': 'tuv',

'9': 'wxyz',

}

def backtrack(idx, comb):

if idx == len(digits):

res.append(comb[:])

return

for letter in digit\_to\_letters[digits[idx]]:

backtrack(idx + 1, comb + letter)

res = []

backtrack(0, "")

return res

18. 4Sum Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: ● 0 <= a, b, c, d < n ● a, b, c, and d are distinct. ● nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order.

def fourSum(self, nums, target):

nums.sort()

results = []

self.findNsum(nums, target, 4, [], results)

return results

def findNsum(self, nums, target, N, result, results):

if len(nums) < N or N < 2: return

# solve 2-sum

if N == 2:

l,r = 0,len(nums)-1

while l < r:

if nums[l] + nums[r] == target:

results.append(result + [nums[l], nums[r]])

l += 1

r -= 1

while l < r and nums[l] == nums[l - 1]:

l += 1

while r > l and nums[r] == nums[r + 1]:

r -= 1

elif nums[l] + nums[r] < target:

l += 1

else:

r -= 1

else:

for i in range(0, len(nums)-N+1): # careful about range

if target < nums[i]\*N or target > nums[-1]\*N: # take advantages of sorted list

break

if i == 0 or i > 0 and nums[i-1] != nums[i]: # recursively reduce N

self.findNsum(nums[i+1:], target-nums[i], N-1, result+[nums[i]], results)

return

19. Remove Nth Node From End of List Given the head of a linked list, remove the nth node from the end of the list and return its head

class Solution:

def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:

res = ListNode(0, head)

dummy = res

for \_ in range(n):

head = head.next

while head:

head = head.next

dummy = dummy.next

dummy.next = dummy.next.next

return res.next

20. Valid Parentheses Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type

class Solution:

def isValid(self, s: str) -> bool:

valid\_brack = [('{', '}'), ('(', ')'), ('[', ']')]

stack = []

for c in s:

if len(stack)>0 and (stack[-1], c) in valid\_brack:

stack.pop()

else:

stack.append(c)

return len(stack)==0